IB Paper 8: Photo Editing

Lecture 3: Morphing and Colour manipulation

N G Kingsbury (given by J Lasenby in 2016)

Signal Processing Group,
Engineering Department,
Cambridge, UK

Easter 2016

Morphing the image

- Morphing of an image implies spatial distortion of parts of the image.
- May be familiar with full morphing which gradually converts one image into another via a series of local translations and mixings.
- Full morphing is complex – so here we just do some partial morphing which consists of local translations. Nevertheless leading to some amusing effects!
- `interp2()` is used once again to achieve smooth translations. A simple user-interface is also far from straightforward.

The basic script: `ph_morph`

- As for previous scripts, `ph_morph` contains cases which are selected by **mode** – however, this is more complicated than previous scripts and has 11 cases.
- morphing is defined by control points stored in a $n \times 5$ matrix cp. Each row contains start point, end point and morph radius.
- Control points are selected by mouse clicks.
- Morphing shifts pixels near the start point $\mathbf{c}_0 = [u, v]$ to be near the end point $\mathbf{c}_1 = [p, q]$. Radius r tells us how large a region around the start point is shifted.
- A Gaussian blob $g(s, t)$, centred on $\mathbf{c}_1$, with standard deviation r, defines our shift.

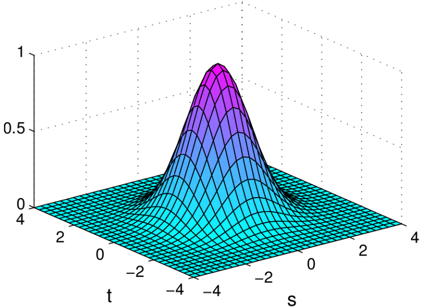
The morphing process

A field of shift vectors is therefore given by

$$\mathbf{x}_{shift}(s,t) = g(s,t)[u-p,\ \ v-q]$$

$$\text{where}\quad g(s,t) = \exp\left(-\frac{(s-p)^2+(t-q)^2}{2r^2}\right)$$

Hence the pixel at a given location $[s,t]$ in the output image will be interpolated from location $[s,t]+\mathbf{x}_{shift}(s,t)$ in the input image.



The morphing process cont...

- For the `interp2` function, we need to specify vectors of resampling points, $\mathbf{r}_i$ and $\mathbf{c}_i$ – these must therefore be monotonically increasing.
- As we are adding $\mathbf{x}_{shift}$ to something with gradient $+1$ (e.g. $1..n, 1..m$), need a gradient of greater than -1, so that overall the gradient is still positive

$$\frac{d}{dx}\, e^{-x^2/2r^2} = -\frac{x}{r^2}\, e^{-x^2/2r^2} \quad \text{and} \quad \frac{d^2}{dx^2}\, e^{-x^2/2r^2} = -\frac{1}{r^2}\left(1 - \frac{x^2}{r^2}\right) e^{-x^2/2r^2}$$

Steepest gradient: $x = \pm r$ and is $e^{-0.5}/r = 0.6065/r$ in magnitude. Scale this by the displacement vector $\mathbf{d} = [u - p,\ v - q],\ \implies\ r > 0.6065|\mathbf{d}|$ to avoid a negative overall gradient.

- In practice we ask that $r \geq 0.8\sqrt{(u-p)^2 + (v-q)^2}$

ph_morph in more detail...

- **Init**: opens the command window, with buttons, slider etc.
- **New**: resets variables and sets mode to **Get points**
- **Add**: Same as New but does not reset variables
- **Delete**: Removes final row of control points and updates
- **Play**: plays a sequence of 10 'interpolated' morph frames
- **Load**: loads control points from a user-selected file
- **Save**: saves control points in a user-selected file
- **Select frame**: displays a frame in the play sequence
- **Enter frame**: enter frame to display numerically
- **Get points**: allows entry of control points via cursor
- **Close**: closes morph box and updates images

The function im_morph

- `im_morph` calculates the shift field, `xshift` and the morphed output image `yui` from input image and control points.
- A for loop computes the gaussian blob $g(s,t)$ for each set of control points – $g(s,t) * (\mathbf{c}_1 - \mathbf{c}_0)$ is then added to the old `xshift`.
- Note that in the code the x and y components of `xshift` are represented as real and imaginary parts of a complex matrix
- We then calculate row and column indices for interpolation, $\mathbf{r}_i$ and $\mathbf{c}_i$, and `interp2` is called.

Colour conversions and colour correction ph_colourshift

- Colour is a difficult field! Partly because much of it is concerned with human perception.
- The script `ph_colourshift` is concerned with adjusting colours within the image.
- This is a general interface to allow each colour component to be scaled or shifted.
- To increase the effects we can produce we can work in a number of colour spaces: RGB, YUV and HSV. We first look at and understand each of these spaces.

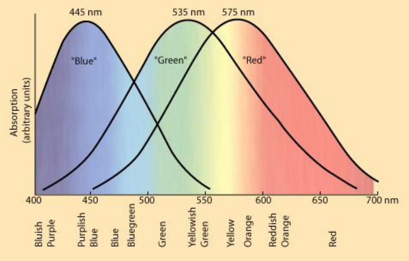
Overview: RGB, YUV and HSV

RGB (*Red Green Blue*) is the normal colour space used for most modern video display cards. The two others are YUV (*Luminance, Chrominance*) and HSV (*Hue, Saturation, Value*).
Colours and RGB, YUV and HSV values are below:

Colour	*R*	*G*	*B*	*Y*	*U*	*V*	*H*	*S*	*V*
Black	0	0	0	0	0	0	-	-	0
Mid-grey	0.5	0.5	0.5	0.5	0	0	-	0	0.5
White	1	1	1	1	0	0	-	0	1
Red	1	0	0	0.3	−0.15	0.4375	0	1	1
Yellow	1	1	0	0.9	−0.45	0.0625	0.167	1	1
Green	0	1	0	0.6	−0.3	−0.375	0.333	1	1
Cyan	0	1	1	0.7	0.15	−0.4375	0.5	1	1
Blue	0	0	1	0.1	0.45	−0.0625	0.667	1	1
Magenta	1	0	1	0.4	0.3	0.375	0.833	1	1
Pink	1	0.5	0.5	0.65	−0.0750	0.2188	0	0.5	1
Pale green	0.5	1	0.5	0.8	−0.15	−0.1875	0.333	0.5	1
Pale blue	0.5	0.5	1	0.55	0.225	−0.0313	0.667	0.5	1

The RGB Colour Space

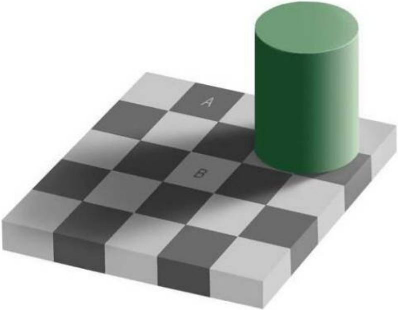
- They eye has 2 types of photoreceptors, rods and cones. Only the cones are sensitive to colour.
- Experiment suggests that there are 3 types of cone. We therefore take colour space to be 3D.



As the 3 types of cones peak in sensitivity roughly at the red, green and blue wavelengths, it is natural that our basic 'colour space'/'coordinate system' for graphics should be RGB

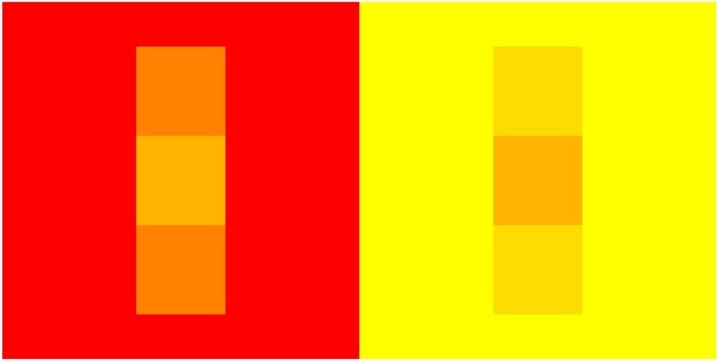
RGB values range from 0 to 1 – then scaled up by 255 to cover (`uint8`) 0 to 255. Problem with RGB is that each colour affects apparent brightness (luminance) by different amounts. Eye is most sensitive to changes in luminance.

Colour Perception and Contrast



The eye is very insensitive to small spatial changes in brightness.
Therefore contrast is hugely important to our perception of colour.
Square A is the same 'colour' as square B.

Colour Perception and Contrast



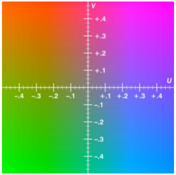
The YUV Colour Space

- YUV has been the colour encoding system used for analogue television worldwide (PAL, NTSC, SECAM standards). Y is a luminance component and U,V are chrominance components. There is a very simple conversion from RGB to YUV:

$$Y = 0.3R + 0.6G + 0.1B$$

$$U = 0.5(B - Y) = -0.15R - 0.3G + 0.45B$$

$$V = 0.625(R - Y) = 0.4375R - 0.375G - 0.0625B$$



- (Note: other places use $V = 0.877(R - Y)$). U and V indicate how far away from grey the colour is in the red and blue directions.
- 0.5 and 0.625 are the largest multiples of $\frac{1}{8}$ which keep U and V in the range ± 0.5.

YUV cont...

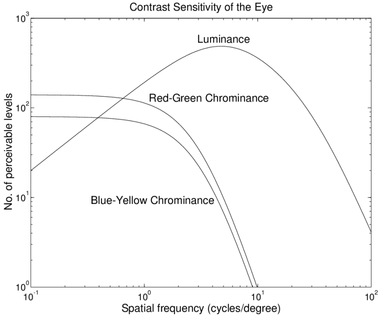
- RGB$\longleftrightarrow$ YUV transformations can easily be performed in hardware.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \mathbf{C} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{where} \quad \mathbf{C} = \begin{bmatrix} 0.3 & 0.6 & 0.1 \\ -0.15 & -0.3 & 0.45 \\ 0.4375 & -0.375 & -0.0625 \end{bmatrix}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \mathbf{C}^{-1} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \quad \text{where} \quad \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 1.6 \\ 1 & -0.3333 & -0.8 \\ 1 & 2 & 0 \end{bmatrix}$$

Sensitivity to YUV



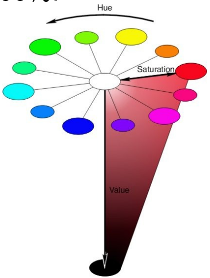
- Max sensitivity to Y occurs around 5 cyc/deg – i.e. stripes with width of about 1.8mm at a distance of 1m.
- x-axis is the frequency of an alternating pattern of parallel stripes with sinusoidally varying intensity.

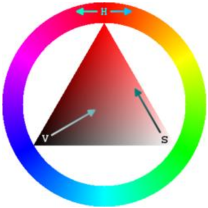
Sensitivity to YUV cont...

- Almost no response above 100 cyc/deg: [0.1mm at 1m]. Typical screen has diagonal of about 480mm – for resolution of 1280×1024 pixel edge size is about 0.29mm.
- Sensitivity to Y drops off at low spatial frequencies – we are not good at estimating absolute luminance (if no temporal change).
- Max chrominance sensitivity occurs at much lower spatial frequencies. Max sensitivity of U is about $\frac{1}{6}$ that of Y and about $\frac{1}{2}$ that of V.
- Chrominance sensitivity falls off rapidly above 1 cyc/deg.
- YUV therefore good for compression and transmission – as we can sample U and V at a lower rate and quantise them more coarsely. [NB JPEG and MPEG].

The HSV Colour Space

- Hue = colour type: standard range from 0 to 360: see wheel/hexacone.
- Saturation = intensity of colour: ranges from 0 to 100%. Measures the dominance of the hue.
- Value = brightness or intensity of the light: ranges from 0 to 100%.





- HSV popular in graphics applications for choosing colour (via wheel) of elements. (Note that above scales can change in applications – eg Paint: H,S,V go from 0 to 240).

HSV in more detail

$$V = \max(R, G, B) \qquad \text{and} \qquad S = \frac{V - \min(R, G, B)}{V}$$

- So that V represents the approximate luminosity and S is the approximate 'distance' of the colour from some shade of grey.

$$H = \begin{cases} \frac{G-B}{6SV} & \text{if} \quad V = R \quad \text{and} \quad G \geq B \\ \frac{B-R}{6SV} + \frac{1}{3} & \text{if} \quad V = G \\ \frac{R-G}{6SV} + \frac{2}{3} & \text{if} \quad V = B \\ \frac{G-B}{6SV} + 1 & \text{if} \quad V = R \quad \text{and} \quad G < B \end{cases}$$

- Take, for example, pink $= [R = 1,\ G = 0.5,\ B = 0.5]$, we see from above that $H = 0$ (as for red) but $S = 0.5$, ie half that of red.
- H and S are essentially a polar coordinate representation of the chrominance (see wheel/cone)

The script ph_colourshift

- **Init**: Sets up command box and initialises variables.
- **Slider**: called when any of the sliders are activated
- **Edit box**: called when boxes are used to edit variables
- **Reset**: sets all gain components to 1 and offset components to 0.
- **Swap**: used to move between, *RGB*, *YUV* and *HSV*
- **Close**: closes command box and redisplays images
- **Colour**: called when **mode** is set to **Colour**. This performs the colour correction/conversion – final result converted back to RGB for display.

Colour mode and conversion functions

- In RGB mode, gain and offset corrections applied directly to each colour slice of input image `xui`, e.g.

```
yui(:,:,k) = uint8(cgain(k)*double(xui(:,:,k)) +
                   cofst(k)*256);
```

  `uint8` ensures that any out of range values are set to 0 or 255.

- In YUV mode gain and offset corrections are applied to `xyuv`, which is `xui` converted to YUV.

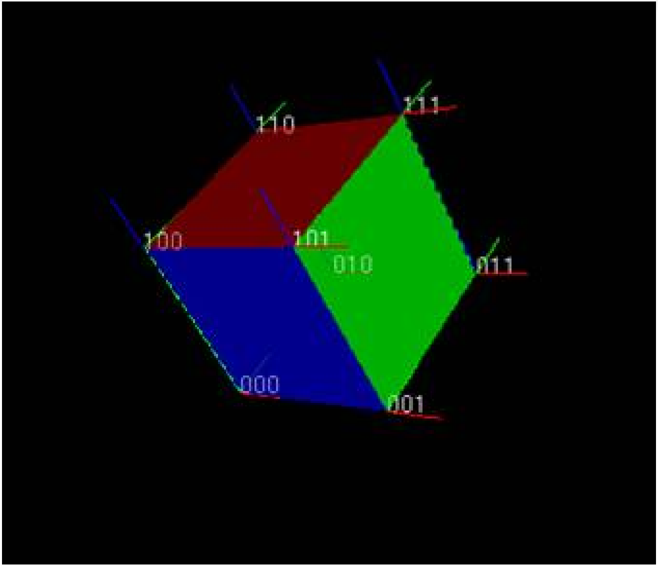
- In HSV mode gain and offset corrections are applied to `xhsv`, which is `xui` converted to HSV.

- Note: the following functions convert between formats: `rgb2yuv`, `rgb2hsv`, `yuv2rgb`, `hsv2rgb`. These are mostly straightforward, excepting `hsv2rgb` which is more complicated due to the non-linear nature of the transform.

Summary

- Section 5 of the notes outlines how the Photo Editor morphs images using user-defined control points and interpolation.
- Section 6 deals with the somewhat complicated issues surrounding colour. The three most common colour spaces and altering aspects of image colour are discussed.

J. Lasenby (Easter 2016)


110
111
100
101
010
011
000
001